— Raja R.K
The author is a Lead Engineer with HCL Technologies, Chennai. He can be contacted at: rajark_hcl@yahoo.co.in.

Techknow Grid	
Language	Java
Platform	Independent
Level	Advanced

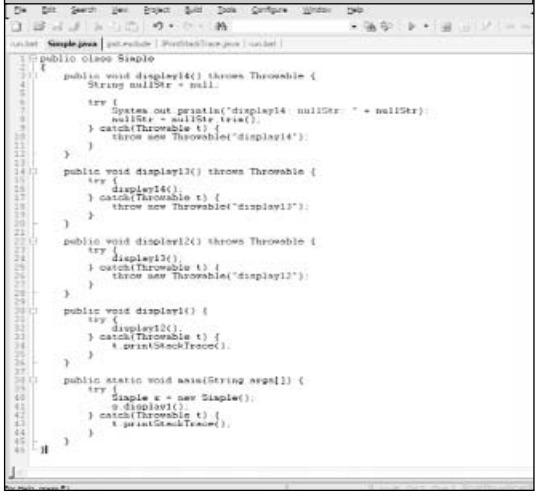
JPrintStackTrace

The traditional Java stack trace may not be all that helpful in your debugging efforts. The author introduces you to a solution for such cases.

Many times you may end up in a situation where the usual Java stack trace is of no use for you to debug the problem. The stack trace may not even show you the correct source code line that triggered the problem. Also, the program you are trying to debug could be a legacy one or you may not have access to the source code or even if you have the source code, recompiling and building them may not be easy. In such cases, you may find the need for a tool that can enhance the Java stack trace and help you easily debug and fix the problem, imperative.

In this article, we shall get to know a tool called JPrintStackTrace (developed by the author) that can be employed for a more precise Java stack trace without making any changes to the source code.

Fig 1 Simple.java

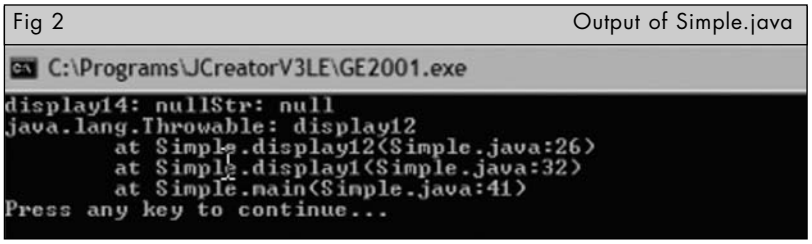


```
1  public class Simple
2  {
3      public void display14() throws Throwable {
4          String nullStr = null;
5
6          try {
7              System.out.println("display14 nullStr " + nullStr);
8              nullStr = nullStr.trim();
9          } catch(Throwable t) {
10             throw new Throwable("display14");
11         }
12     }
13
14     public void display13() throws Throwable {
15         try {
16             display14();
17         } catch(Throwable t) {
18             throw new Throwable("display13");
19         }
20     }
21
22     public void display12() throws Throwable {
23         try {
24             display13();
25         } catch(Throwable t) {
26             throw new Throwable("display12");
27         }
28     }
29
30     public void display1() {
31         try {
32             display12();
33         } catch(Throwable t) {
34             t.printStackTrace();
35         }
36     }
37
38     public static void main(String args[]) {
39         try {
40             Simple s = new Simple();
41             s.display1();
42         } catch(Throwable t) {
43             t.printStackTrace();
44         }
45     }
46 }
```

JPrintStackTrace is Java Platform Debugger Architecture (JPDA) based tool. It runs on a separate JVM and can connect to your program for enhancing Java stack trace. Since it works at the JVM level, you don't need to make any code changes to your program. The power of JPrintStackTrace tool could be best explained with an example. Consider the Simple.java program shown in **figure 1**.

This program tries to dereference a null string at line 8 in function

Fig 2 Output of Simple.java



display14(). Looking carefully at the sample program you will observe that the exceptions are catched and a new exception object is thrown to the caller and this continues till the function display1() where it is displayed to the user.

Now let us compile and run the program.

From **figure 2** you can clearly see that the Java stack trace is of no use. It does not provide the correct source code line that triggered the exception. You now have to either change the source code to handle the exceptions correctly or should run the program in a step-by-step mode in a debugger to identify the root cause. However, both these approaches are either time consuming or not feasible for several reasons, as described earlier.

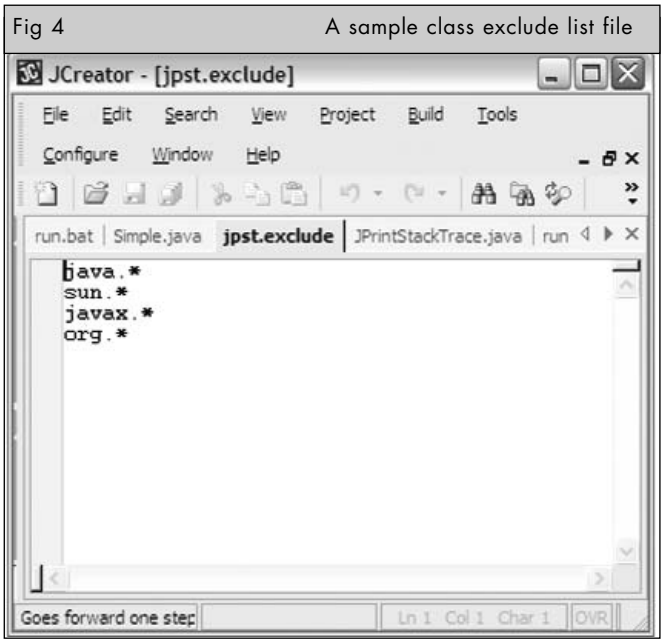
We shall now run the same Simple.java under JPrintStackTrace tool. First compile and run the Simple.java in debug mode as shown in **figure 3**.

From figure 3 you can see that the Simple program is started in debug mode and listens at port 4000.

You can get JPrintStackTrace usage help by running the tool without passing any command line options. It takes four command line options as described below:

1. -r <remote_host_name>: The name of the machine running the program to be debugged. In our case it is 'localhost';
2. -p <port_no>: The port number at which the program to be debugged is listening. In this case it is '4000';
3. -e <class_exclude_list_file>: List of java classes that need to be excluded by the tool. A sample class exclude list is shown in **figure 4**; and
4. -l <log_file>: Log file.

Fig 4 A sample class exclude list file



Now let us start the JPrintStackTrace tool and connect to Simple program as shown in **figure 5**.

Once connected, the JPrintStackTrace tool will run the Simple program and listens for exception event in target JVM. Upon getting the exception event, it will call the printStackTrace() method on the exception object on the target JVM to get a detailed stack trace as shown in **figure 6**.

Fig 5 Running JprintStackTrace

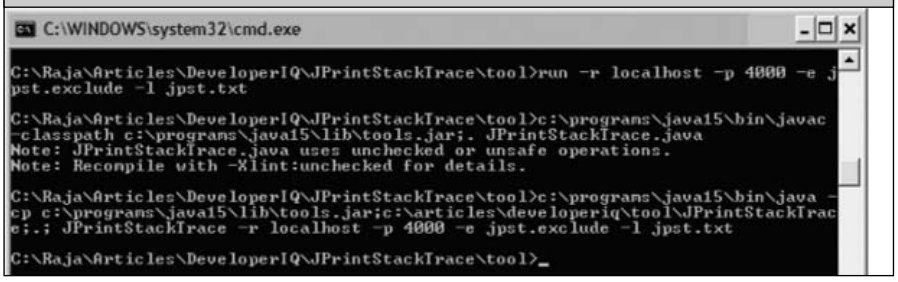
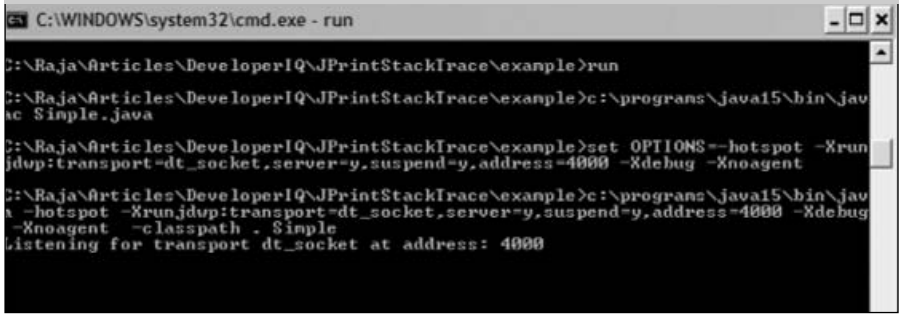
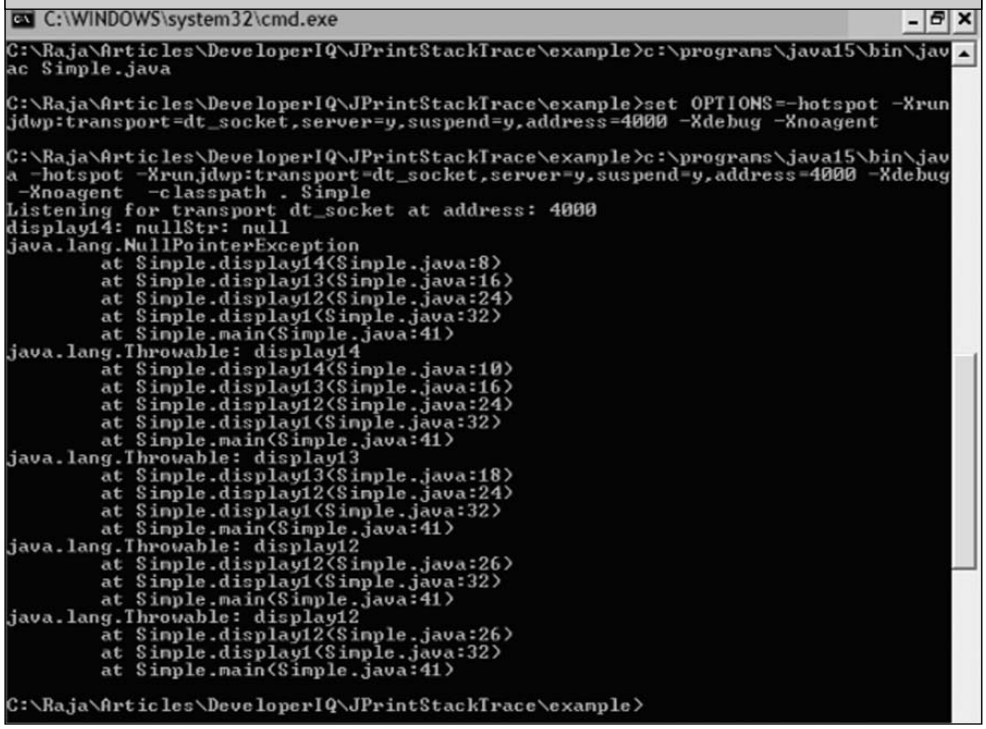


Fig 3 Running Simple.java in debug mode



From figure 6 you can clearly see the source code line that triggered the exception. It is line 8 in function display14()! On comparing the outputs shown in figure 2 and figure 6, you will definitely appreciate the power of JPrintStackTrace tool. Remember, you get all this without making any changes to your source code!

Fig 6 JprintStackTrace in action!!!



Source code for JprintStackTrace is available in CD2. Happy debugging!!!

DIQ

Reference:

http://www.java.sun.com/products/jpda/index.jsp

The author is working at HCL Technologies (Networking Product Division), Chennai. He can be contacted at: rajark_hcl@yahoo.co.in.

Nokia in Eclipse Foundation

Nokia and the Eclipse Foundation have announced that Nokia has joined the Eclipse Foundation as a Strategic Developer and Board member. Nokia will support the work of the Eclipse Open Source community by contributing software and developers to a proposed new Eclipse project.

As a Strategic Developer in the Eclipse Foundation, Nokia will lead a project to create a framework for mobile Java developer tools, including complete tooling support for J2ME (Java 2 Micro Edition). The project will deliver a sustainable mobile tools offering for all developers and companies who wish to create mobile Java applications and build commercial tools for Java. Nokia plans to donate several components of its existing Java development tools technology as well as actively develop new software to introduce tools for the creation of both Mobile Information Device Profile (MIDP) and Connected Device Configuration (CDC) based mobile Java applications. Furthermore, Nokia plans to use the Eclipse tools platform widely in its tools portfolio and will actively contribute to several existing Eclipse projects beyond the scope of Java.

"By working closely with Eclipse and proposing a new Open Source mobile development tools project, we will provide more than two million registered developers in our Forum Nokia program with complete integrated tool packages optimized for Nokia platforms," said Pertti Korhonen, executive vice president and chief technology officer, Nokia. "In addition, developers creating mobile applications will be able to work more efficiently and achieve greater productivity by using the same Integrated Development Environment (IDE) for multiple programming languages and software platforms."

"Nokia's membership in Eclipse reinforces our commitment to Open Source initiatives," Mr. Korhonen added. "This important step enables us to simplify our creation of tools and further harmonize our development tools offering across all Nokia software platforms. We expect to benefit from Open Source innovation and create interest and active participation in Nokia-led mobile tools projects for the benefit of all members of the Eclipse community."

Nokia's Eclipse initiatives date from 2004, when it announced support for the Eclipse platform in the Nokia Developer's Suite for J2ME and the Nokia Mobile Server Services SDK. Also starting in 2004, Nokia contributed key development resources and leadership to the Embedded Rich Client Platform (eRCP) project at Eclipse.